HEPCloud, a new paradigm for HEP facilities: CMS Amazon Web Services Investigation

L Bauerdick[1,2], B Bockelman[1], D Dykstra[1,2], I Fisk[1], S Fuess[2], G Garzoglio[2], M Girone[1], O Gutsche[1,2], B Holzman[1,2,†], H Kim[2], R Kennedy[2], D Hufnagel[1,2], N Magini[1,2], D Mason[1,2], P Spentzouris[2], A Tiradani[1,2], S Timm[2] and E Vaandering[1,2]

[1] CMS

[2] Fermilab

[†] Corresponding author: *burt@fnal.gov*

Preprint: FERMILAB-PUB-16-170-CD

1. Overview

Every stage of a modern HEP experiment requires massive computing resources (compute, storage, networking). Detector and simulation-generated data have to be processed and associated with auxiliary detector and beam information to generate physics objects, which are then stored and made available to the experimenters for analysis. In the current computing paradigm, the facilities that provide the necessary resources utilize distributed High Throughput Computing (HTC), with global workflow, scheduling, and data management, enabled by high-performance networks. The computing resources in these facilities are either owned by an experiment (e.g. at the LHC) or deployed for a specific program (e.g. Intensity Frontier experiments at Fermilab).

The HEP investment to deploy and operate these resources is significant: for example, the current size of the worldwide computing infrastructure for the CMS experiment is 150K cores, with US CMS deploying 15,000 cores at the Fermilab Tier-1 site, and more than 45,000 cores at seven Tier-2 sites. One important aspect of the utilization of compute resources is that it comes in "bursts," production processing being more regular and analysis much less predictable.

The evolution of the HEP experimental program (upgrades, new experiments) will generate increased computing needs that are expected to go well beyond any capacity increases expected from Moore's law or advancements in computing architectures. Additionally, due to power and cooling requirements, new architectures have lower performance per core. By 2025, the muon program, the long-baseline and short-baseline neutrino programs, and the LHC will reach full operational strength as two new programs (High-Luminosity LHC, DUNE) come on-line. The increased precision, event complexity, and luminosity of the HL-LHC alone will push computing needs nearly two orders of magnitude above current HEP capabilities, while generating exabytes of data.

It is essential for HEP to develop the concepts and deploy the infrastructure that will enable analysis of these vast amounts of data efficiently and cost-effectively. The industry trend is to use "Infrastructure as a Service" through Cloud computing, to reduce cost of provisioning and system operations, to provide redundancy and fail-over, to rapidly expand and contract resources (elasticity), and to pay only for the resources needed/used. Following the same paradigm, US HEP facilities could incorporate and manage "rental" resources, achieving "elasticity" that satisfies demand peaks without over-provisioning local resources.

The HEPCloud Facility concept is envisioned to be a portal to an ecosystem of computing resources—commercial or academic—that will help our facilities move away from standalone, siloed solutions. It will provide "complete solutions" to all users, with agreed-upon levels of service, routing user workflows to local ("owned") or remote ("rental") resources based on efficiency, cost, workflow requirements and the policies of the facilities hosting the resources. This will be done transparently to the users, utilizing

a sophisticated decision engine and cost model, and incorporating the necessary policies and infrastructure to manage allocations to potential computing resources (local or remote). The portal concept could provide the means for all laboratories to provide shared resources in the ecosystem, resulting in a large pool of offerings for compute, data archival capabilities, database services, data management, etc., potentially linking all US HEP computing.

In order to investigate the merit of this approach, we established the Fermilab HEPCloud project. The goal is to integrate rented resources into the current Fermilab computing facility in a manner transparent to the user. The first type of external resources considered was commercial clouds, partnering with Amazon Web Services as the provider. For our studies, we identified use cases that both demonstrate the necessary aspects of the concept, and that are also useful to the experimenters. One of the use cases focused on CMS Monte Carlo generation and reconstruction, targeting physics results for the Moriond conference in March 2016. This use case studied the scalability and sustainability of elastic provisioning of AWS resources through the portal, and exercised the prototype decision engine and cost model.

2. CMS Use Case: Introduction

The CMS experiment is confronting a large and ever-increasing computing challenge, even before the start of the HL-LHC. To meet the growing computing needs, CMS has investigated use of resources beyond the traditional grid-provided systems. One potential area of growth is in dynamically provisioned resources—either via academic and opportunistic access, or through commercially provided computing services. Beginning in 2015, CMS began to seriously explore commercial cloud provisioned resources. The logical platform choice was the market leader [1] in Cloud Infrastructure-as-a-Service, Amazon Web Services (AWS).

In the past, CMS has demonstrated small-scale cloud computing for a short amount of time as a proof-of-concept to investigate feasibility. This demonstration does the next big step and is intended to show the ability to increase the global processing capacity of CMS by a significant fraction for an extended period. Importantly the test was also intended to deliver useful simulated physics events to the collaboration for analysis at a production scale.

CMS was awarded a 9 to 1 matching grant from AWS that allowed the purchase of $300k of credits for computing, storage, and network charges for an investment of $30k. The size of the award was based on an estimate of what it would cost to do one month of large-scale processing. Additionally, a conditional cost waiver was granted for exporting data; as long as the export costs remained under 15% of the total monthly bill, and were transmitted across research networks such as ESNet, the export charges would be waived entirely. This discount program was so successful that it has been extended to researchers at all academic and research institutions. [2]

In this report, we will discuss the tests performed, the services required, and

the scale and performance achieved. We will evaluate the cost to provide dedicated computing resources at Fermilab versus the costs paid to AWS for the same capacity. This report may serve as a source for material for future publications, and the conclusions may be useful for resource planning, but this document is intended as a CMS and Fermilab internal note and the conclusions and discussions are intended for CMS and Fermilab audiences.

3. Workload and Workflows

CMS evaluated several production workflows based on a variety of criteria: physics value and current needs of the experiment, difficulty of the workflow, the potential for gaining knowledge by operating cloud provisioned resources, and the most effective use of the resources (given the potential for additional charges for staging out data). The four workflows considered were: GEN-SIM, Data-RECO, DIGI-RECO, and GEN-SIM-DIGI-RECO. Generation and Simulation (GEN-SIM) is the simplest because it requires only parameters as input, and it runs CPU-bound on a single core. As the simplest workflow, there is not much to learn from it and CMS did not have a need for a large number of new GEN-SIM events, so this workflow was not favored. Data reconstruction (Data-RECO) relies on raw events served by a global data federation and produces output where every job must be completed to maintain the integrity of the dataset. CMS did not have an urgent need for Data-RECO, so also this workflow was not favored. Event crossing simulation and reconstruction (DIGI-RECO) adds several challenging elements: the input data from the previous step must be served over the global data federation, but the additional minimum bias events to create the full crossing need to be served from local storage in the CMS "classic" mixing scenario. Each event reads hundreds of megabytes of minimum bias "pile-up data" and reading the events over the wide area would result in very low CPU efficiency. The DIGI-RECO workflow was a consideration because of the technical capability that could be demonstrated. In the end, the workflow that was used for the majority of the processing was the workflow that combined *all* the elements into a single chain: GEN-SIM-DIGI-RECO. It is slightly simpler than DIGI-RECO, because it requires only parameters as input, but it has a higher amount of processing per event because it includes the generation and simulation steps. The higher amount of CPU results in a more favorable ratio of processing to final event size, which allowed CMS to remain within the 15% data egress waiver. GEN-SIM-DIGI-RECO is a single complete workflow and several high impact samples were identified to demonstrate the value of the AWS resources.

The individual pieces of GEN-SIM-DIGI-RECO refer to the four steps in jobs themselves, each an invocation of the CMS Software framework (CMSSW) processing the results of the step prior to it:

(i) GEN: Madgraph 4-vector event generation

(ii) SIM: Propagation of particles through the GEANT detector simulation

(iii) DIGI: Electronics simulation and event digitization, including mixing in pile-up data prestaged to S3 storage

(iv) RECO: Reconstruction of the data into physics quantities used in analysis

Only the products from the last step were kept and staged out to the Fermilab EOS storage system, namely the AoD and the mini-AoD. In past production campaigns, CMS typically executed GEN, SIM, and DIGI-RECO as separate sets of jobs. Each job set involves stage-out of the intermediate data, sometimes restaging from tape—I/O activities which would increase the production costs at a resource like AWS. Since all the CMS Software framework releases in question were in production and available, work was put into deploying these steps all into a linked chain of jobs on the workers themselves, called a step-chain.

For the large-scale run, four large Standard Model background samples were produced in full chain GEN-SIM-DIGI-RECO workflows—abbreviations for each in parentheses used later on:

- Drell-Yan+Jets to dilepton for mass ranges of 10–50 GeV (DY_M10-50)
- Drell-Yan+Jets to dilepton for mass > 50 GeV (DY_M50)
- T-tbar to jets (TTJets)
- W+Jets to lepton/neutrino (WJetstoLNu)

4. Procurement Evaluation

The Fermilab HEPCloud team, in consultation with CMS staff and the Fermilab Procurement office, wrote a set of specifications for commercial cloud providers. This included a set of financial and technical requirements to satisfy the need of the Fermilab HEPCloud Facility project. The financial requirements included the ability to track spending by groups, to account for regular spending and credits, to access technical support, and to pay in advance with pro forma invoicing. The technical requirements included access to a minimum scale of storage and CPU cores, guaranteed network bandwidth to the ESNet science network, support for certain API's to launch virtual machines, support for monitoring, and the ability to alert on preset levels of spending. A Statement of Work was prepared that described the activity of CMS and three other projects, and specified a total amount of computing services that would be purchased. A wide variety of cloud providers and resellers were requested to bid on this Request For Quotations (RFQ) [3]. The bids were evaluated according to a predetermined set of best value criteria. Once the qualifying bid was selected, the team certified that the services of the selected provider met the requirements of the RFQ. DLT Solutions, a reseller of Amazon, was awarded the contract.

5. Services

5.1. Services Deployed in AWS

5.1.1. Squid and Frontier The sole on-demand service deployed for the CMS run is a dynamically scaled web cache—a Squid service that is used to cache CVMFS code and Frontier database queries. From previous studies[5] , it was observed that a squid service that was not co-located with compute nodes had a latency that was high enough to degrade performance. In order to instantiate the service, AWS CloudFormation[1] is used to orchestrate the entire process for launching and tearing down the service and dependent infrastructure. The following services and/or components are configured by the the CloudFormation template created by the Fermilab HEPCloud team:

- Elastic Load Balancer (ELB)[2]
- Auto Scaling Group[3] (from 1 to N servers) and policies
- Squid-optimized Amazon Machine Image
- Route 53[4] DNS CNAME Record Set
- CloudWatch[5] alarms

One CloudFormation stack as described above is deployed per availability zone. Each stack has a known, fixed address, which is pre-determined by the Route 53 service. An initialization script runs in each virtual machine at launch that auto-detects which region and availability zone it is in, as well as the public hostname of the machine. CMS-specific configuration files (a CVMFS localization file, site-local-config.xml, storage.xml) are modified at launch to point to the appropriate squid server and the location of the pile-up data via an AWS Simple Storage Service (S3)[6] URI.

The squid server is used to cache both the experiment-specific software, which is delivered via CVMFS, and event-specific database information, which comes from Frontier. CVMFS maintains its own local cache on the local disk, while Frontier does not. In order to efficiently cache the Frontier information, we instantiated a squid cache on each worker node, reducing the net load on the AWS central squid servers and on the main Frontier servers at CERN, as well as the cost of traffic through the Elastic Load Balancer. At peak load, each stack was running eight squid servers, with four squid processes per server. Each server was instantiated on a 4-core c3.xlarge instance. Eight stacks were instantiated, each one corresponding to a different region/zone combination.

5.1.2. AWS Network Configuration Each AWS region has a default Virtual Private Cloud (VPC) that defines the network configuration for an AWS account. The default

[1] https://aws.amazon.com/cloudformation/
[2] https://aws.amazon.com/elasticloadbalancing/
[3] https://aws.amazon.com/autoscaling/
[4] https://aws.amazon.com/route53/
[5] https://aws.amazon.com/cloudwatch/
[6] https://aws.amazon.com/s3/

VPC was modified to accommodate the CMS use case. The VPC is set up with one Internet Gateway configured with a subnet per Availability Zone. Once the VPC is configured, Security Groups act as firewalls to control the traffic allowed to reach the virtual machines. Two Security Groups were configured. One Security Group allowed the squid servers to contact the general internet in order to retrieve data and cache it. The other Security Group restricted outbound network access to Fermilab, CERN, and the AWS public IP addresses; this group allows inbound ssh and ntp access from Fermilab only.

5.1.3. AWS Limits In order to ensure that the AWS service is properly scaled to support the user workflow and to control potential runaway costs, the service has a number of adjustable global default limits, some of which are hidden to the end-user. The HEPCloud team encountered several of these during testing and the initial ramp-up of the CMS workflows.

The Elastic Network Interface (ENI) limit was discovered when the number of instances exceeded the default allocation and new Squid/Frontier instances began to generate errors when attempting to provision additional ENI. The default limit is dynamically generated, but can be statically set upon request; the limit was increased to 5000.

The HEPCloud team was notified ahead of time by AWS that the use case would require an increase in Elastic Block Storage (EBS) limits. We initially expected to use an Amazon Machine Image (AMI) with a single 7 GB EBS volume for the operating system, and two ephemeral disk volumes. After gaining some experience with the AWS service, EBS-only AMIs were added to the list of provisionable resources in order to reduce the overall preemption rate and take advantage of good price-performance. This required an additional EBS limit increase request from 20 TB to 300 TB per region.

The limits governing the number of spot instance requests per region had to be significantly increased over the defaults—from 20 to 5500—in order to scale to 58,000 cores. As a precaution, the limits governing the number of non-spot instances were lowered to 20, the expected bounds of the on-demand Squid/Frontier instances.

Additionally, we had to raise the limit on the number of entries per Security Group from 50 to 125. The project, in fact, established a requirement for a "deny all" security posture in order to reduce the risk of errant jobs running up outbound network costs, as described in the previous section. Since we were accessing S3 over its public network interface, we had to explicitly enable outbound access from our instances to the S3 endpoints in each region. This access was granted by configuring a large number of whitelisted subnets in the Security Groups.

5.1.4. AWS Spot Instances AWS sells their excess resource capacity following a market model called "Spot Market". For every combination of machine type, availability zone, and region, users supply a bid price that represents the maximum that they are willing to pay per hour of computing time. AWS sets a dynamically-changing "spot price"

based on the current supply and demand. If the user's bid price is above the spot price, and there is sufficient capacity in the resource pool, the resources are provisioned at the spot price. If the spot price fluctuates above the bid price after a resource has been provisioned, the user is pre-empted with a two-minute advance notice. Resources are charged on the hour boundary and when the instance is terminated by the user; in the event of preemption, the last fraction of an hour is not charged.

5.2. Services Deployed at Fermilab

5.2.1. Accounting and Billing We wrote a new probe for the OSG Gratia accounting system to collect usage data from Amazon. It polls the AWS monitoring interface every hour to detect the number of machines instantiated by instance type, the associated virtual organization and AWS account, and the spot price charged for that hour. For instances that have been terminated, it records the termination reason and time. This information is recorded in the Gratia database and the Gratiaweb service can be used to plot it.

DLT Solutions supplies a billing summary of usage in comma-separated-value format and provides an hourly report on all AWS service usage. Custom routines were written to parse this billing data and keep track of our balance, so that we could know how much remaining funds were available. We also set up our own alarms to alert if the burn rate was unexpectedly high. Estimates of the data egress cost and its ratio to the total cost were also calculated. Because monthly data egress charges below 15% of the total cost are not charged [2], this estimate informs operations of the potential costs of data egress. An access-restricted Grafana instance was deployed for monitoring financial data.

5.2.2. glideinWMS The glideinWMS workload management system [4] was used to provision the worker nodes used during the CMS run. A development version of glideinWMS was deployed to make available some of the new features needed to run at scale. During the testing period prior to ramp-up, several patches were applied *in situ* to address various issues found. These patches were provided to the glideinWMS development team for addition to later development releases.

The glideinWMS HEPCloud factory was configured with more than 260 entries, consisting of nearly every combination of US region, availability zone, and instance type, and including parameters such as the maximum instance lifetime. The AMI ID is passed to the Factory from the glideinWMS HEPCloud frontend, as are the credentials needed to launch the instances.

The instances are configured to run a bootstrap service. This service parses a base-64 encoded string ("user-data") that is passed to the instance when provisioned. The user-data contains the instance lifecycle parameters, the X.509 proxy used for daemon communication, and the glideinWMS pilot arguments. Among them, there is the URL of the glideinWMS pilot scripts. The bootstrap service downloads the pilot scripts and

associated files, then launches the pilot within the instance.

The lifecycle of the provisioned instance is as follows. The factory requests instances as prompted by the frontend. The maximum lifetime is passed to the instance as part of user-data and enforced by the system. As typical, the pilot exits when no more jobs match the resources or when the maximum lifetime is exceeded. The instance is configured to shut down when the pilot exits. Additionally, an administrator can issue condor_rm commands on the factory to remove provisioned instances.

5.3. Reporting

The CERN Dashboard[7] was used to track all the metrics collected by the workflow. Included in the workflow data are total runtimes for each step, data I/O, and efficiencies.

For infrastructure reporting and monitoring, Fermilab deployed a Grafana instance with customized views. The views include aggregate numbers for AMI types per availability zone and region, number of cores provisioned, and a running count of instances in running, idle, and preempted state.

We also leveraged an already-deployed ELK (ElasticSearch, LogStash, Kibana) stack to read in all the data available from the local HTCondor job scheduler, allowing rich data mining of the HTCondor job data. Additional data were available via logfiles that the jobs stage out as part of their routine workflow.

5.4. Operational Setup

At the time of the workflow testing and execution, the CMS global HTCondor pool was not capable of increasing in scale by an additional 60,000 jobs. To address this, we bypassed the CMS global HTCondor pool and provisioned a separate HEPCloud HTCondor pool. We deployed three independent machines at Fermilab running the WMagent workflow management service as schedulers and two additional nodes to serve as a highly-available HTCondor central manager.

6. Costs and stability of services

Motivated by a previous study [6], we selected a simple bid strategy for spot pricing, which was to bid 25% of the "on-demand" price for a given resource.

In this study, the static bid of 25% was found to perform as well or better than various adaptive algorithms previously described in the literature. In order to "diversify our portfolio" and improve the availability and stability of the system at scale, we bid in more than 100 different spot markets, representing nearly all the regions and zones then available in the US, as illustrated in Figure 1.

The mean lifetime for a provisioned resource was 37.6 hours, while the average job lifetime was 4.7 hours. Figure 2 shows the distribution of provisioned resource lifetimes.

[7] http://dashboard.cern.ch

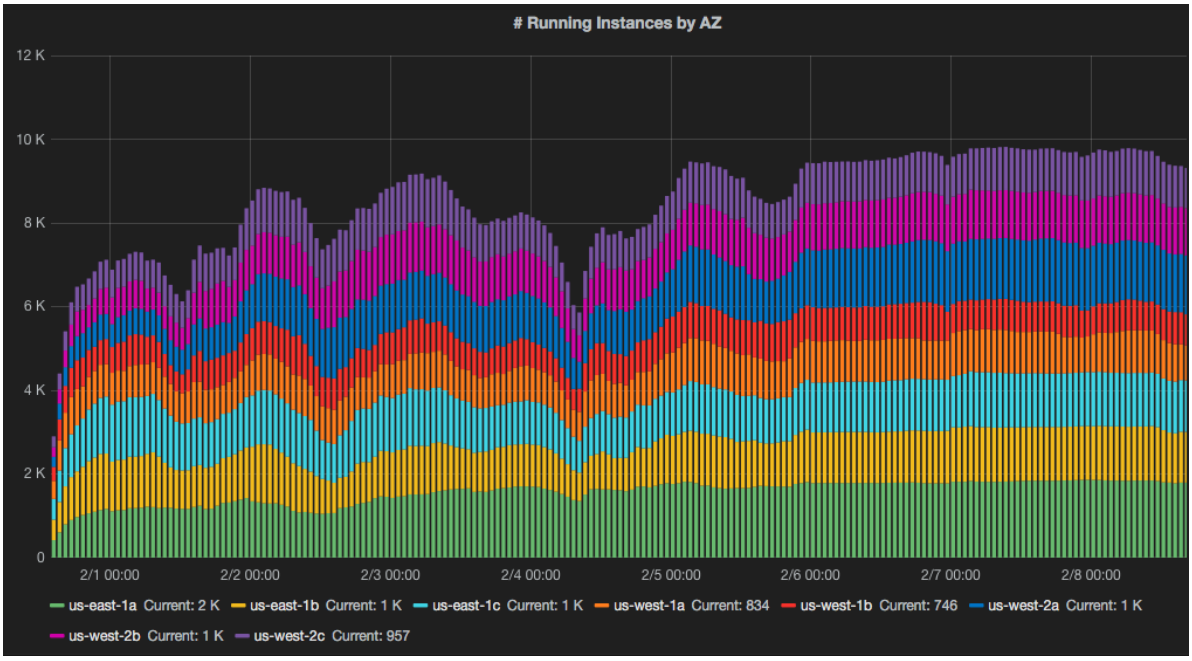

Figure 1: Number of running instances by AWS Region and Availability Zone.

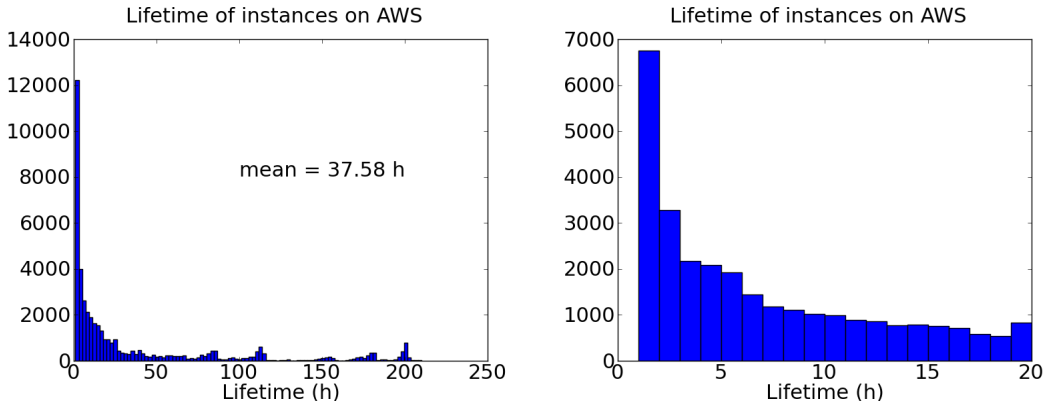

Figure 2: Pilot lifetimes in hours. The left histogram represents the entire distribution; the right histogram is zoomed in to show the distribution for pilots with lifetimes under 20 hours.

While the distribution is peaked in the lowest bin, the tail is very long–some resources remained in the pool for over 200 hours.

Over the course of the 3.2 million job run, 15.5% of the jobs were pre-empted, as shown in Table 1. Preemptions are made visible within HTCondor as the disappearance of a provisioned resource. When a preemption is detected, the scheduler reschedules the job and restarts it on a different available resource. The "number of job starts" is then strictly one less than the number of times a job was preempted. The distribution of this quantity is shown in Figure 3.

There is also a time-dependence to the capability of acquiring resources at scale.

Table 1: Preemption counts for CMS jobs

Number of times preempted	Count	Percentage of total
0	2736240	84.5%
1	403062	12.4%
> 1	101687	3.1%

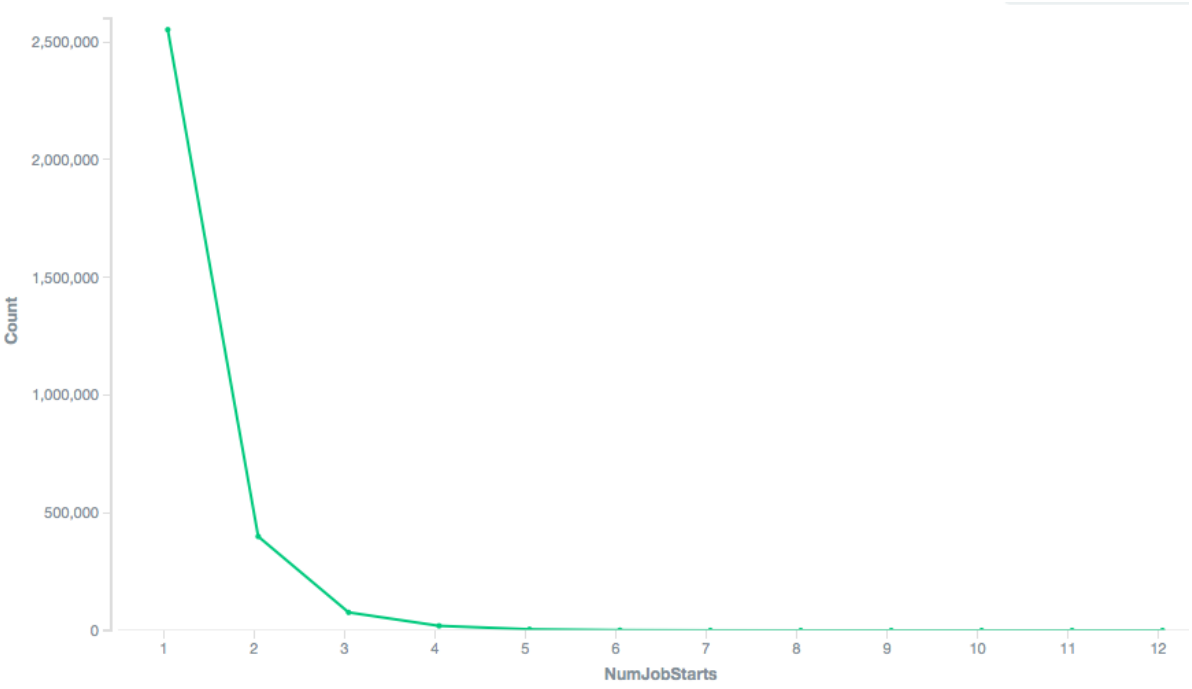

Figure 3: Number of times a job was executed. The peak at 1 corresponds to the large fraction of jobs that were not pre-empted.

During the business day, AWS removes resources from the spot market to fulfill their "reserved" and "on-demand" classes of service. In the late evenings and on the weekends, as the demands on those classes of service go down, the supply of resources into the spot market increases. This is clearly visible in Figure 4—there are peaks in the early morning hours and on weekends, and valleys during the day when resources were removed from pools and machines were being more frequently preempted.

The default strategy used by the glideinWMS frontend and factory was to attempt to distribute the load evenly (on a number of core basis) across nearly all regions, zones and instance types[8]. We found that some region/zone/instance type combinations filled up very quickly and the price quickly moved above our bid price, causing quick pre-emption of those instance types. Over time we ended up accumulating most of the instances which were least likely to get pre-empted. The final instance mix near the end

[8] https://aws.amazon.com/ec2/instance-types/

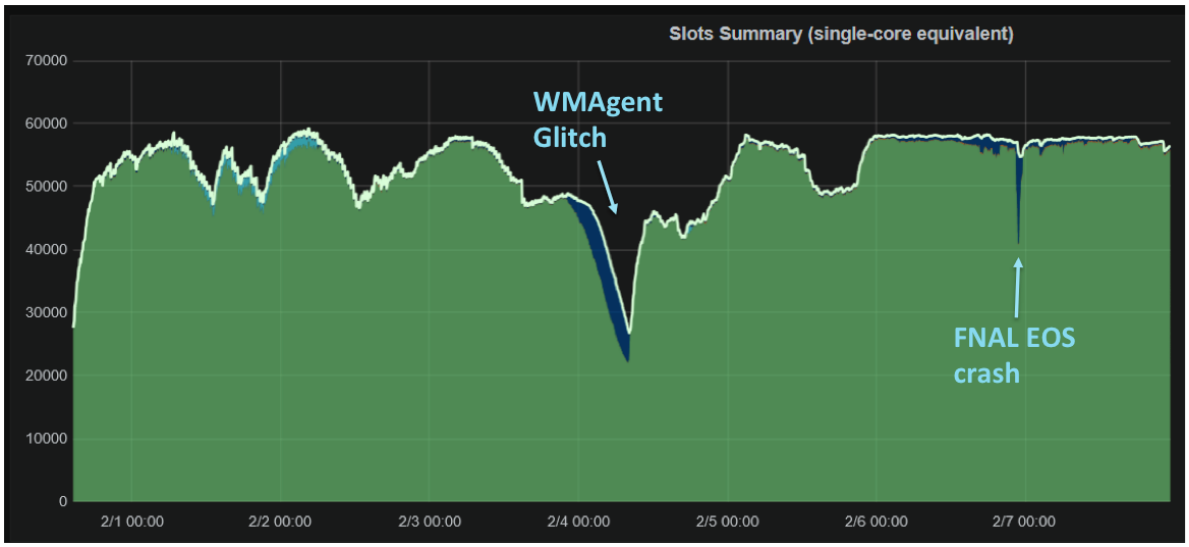

Figure 4: Count of CPU cores on AWS from February 1^{st} (Monday) to February 7^{th} (Sunady), 2016. The plateau near 60,000 cores is limited by the local submission infrastructure. With the exception of the two annotations, the dips are purely due to the dynamics of the spot market.

Table 2: AWS instance type mix at steady state

Instance Type	Number of Instances	Number of Cores per Instance
m3.xlarge	2905	4
m3.2xlarge	1244	8
r3.2xlarge	1109	8
r3.xlarge	826	4
c3.xlarge	759	4
c3.2xlarge	614	8
m4.xlarge	655	4
m4.2xlarge	413	8

of the steady state is shown in Table 2 [9].

During the run, $211,985 was spent on AWS services. 15,085,635 wallclock hours were consumed—giving an average cost per wallclock hour of 1.4 cents. The cost per event for different physics samples is shown in Table 3.

7. Lessons Learned and Operational Considerations

At the beginning of the CMS AWS project, a lot of effort went into looking at the monitoring that would be needed to prevent unnecessary waste of computing resources.

[9] Instance types that provided smaller contributions are not included.

Table 3: Time and cost per job and event, by sample

Sample	Time per job (s)	Time per event (s)	Cost per 100 events
TTJets	25,345	42.2	\$0.016
DY_M10–50	14,111	23.5	\$0.0092
DY_M50	13,214	22.0	\$0.0085
WJetsToLNu	12,235	20.4	\$0.0079

Tracking slow and stuck jobs, rare problems like identifying infinite loops, identifying I/O-bound jobs and other sources of low CPU efficiencies, and protecting against huge log files that would incur high export transfer charges were all considered. On AWS, the financial loss associated with inefficiency is explicit, but an early lesson of the AWS investigation is that we should have the identical monitoring capabilities for dedicated grid resources. On dedicated computing that has been purchased in advance, it is easier to mislead yourself that the inefficiency is not a financial loss, but the issues are the same; the costs have just been amortized up-front and are hidden.

AWS imposes a substantial fee for data egress out of the cloud. Fees begin at \$0.09 per GB and drop to \$0.07 per GB as the total egress per month exceeds 100TB. These charges are waived if the egress charges are less than 15% of the total processing charges [2]. Optimizing the use of resources encourages longer running workflows with small output; there are no charges for import. The CMS choice to use the GEN-SIM-DIGI-RECO with output of only AOD and MiniAOD was partially motivated by the goal of minimizing egress charges. Using AWS for workflows that require large output formats like GEN-SIM, or reconstruction where the large RECO format is needed, would be costly with egress charges.

The CMS "classic" mixing scenario involves opening a file with minimum bias pile-up events and reading a random number of events to simulate a crossing in the DIGI stage. The next event in the job reads a new random number of events. The DIGI step is normally I/O-bound because hundreds of megabytes of I/O are needed for each event that roughly requires 15s of CPU time to process. In order to perform this operation on dedicated sites, we have optimized for I/O, making redundant copies of the pile-up data and ensuring good network connectivity. Serving pile-up over the data federation is not possible at a high scale due to the volume of pile-up involved, so we are constrained to serve locally. When we moved to AWS the initial implementation was to use S3 storage for pile-up, but there is a small charge for each read (seek) operation. We discovered that the CMS application performs so many read operations that the cost of I/O was comparable to the processing time. In order to prevent the cost, we changed the CMS processing model to pull an entire pile-up file to the local worker node storage and read from there. The change increased the job efficiency and reduced the cost.

As discussed above, the system was configured to provision multiple resource types from multiple Amazon regions and availability zones to increase the total scale of

available resources on the spot market. Data, however, is stored at a specific Amazon region. To access the data, therefore, one can opt to store the dataset in one region, for about $0.03 per GB per month, and access it from all regions, incurring in inter-region transfer charges at $0.02 per GB. Alternatively, one can replicate the dataset in all regions, increasing the storage charges, to avoid inter-region transfer charges. The latter was the most cost-effective strategy for the size of the pile-up data.

There are a number of services used by CMS data placement: the PhEDEx service which schedules and logs transfers (including retries); FTS3, which mediates and submits transfers, and the GFAL2 library and utilities, which actually perform the transfers (in this case, using the gridFTP protocol to read data from dCache storage at Fermilab, and the HTTP protocol to write data to S3). While the newest versions of FTS3 and GFAL2 support S3, PhEDEx does not. In order to transfer the pile-up data to S3, we manually generated a list of 11828 files, split it into 10 segments, and submitted them directly to a stand-alone FTS3 server at Fermilab. Retries on failed transfers (at about the 10% level) were then submitted by hand, but made simpler by the FTS client tools' ability to generate lists of failures in a format that could be easily resubmitted to FTS. After a few iterations, the complete dataset of 39.5 TB was transferred to a given S3 region. We enabled monitoring of transfer rates for a single region and found that aggregate throughput was about 400 MB/s.

We also needed to ensure that only authenticated users could read from S3, as we could not control the quantity of data being read (and charged) for unauthenticated access. Read privileges were granted to the compute nodes via "roles" which were assigned at the time of node instantiation. The compute node then retrieves private, public, and session keys via the AWS Security Token Service (STS); this method is preferable to the less-secure method of passing in private keys from outside AWS with appropriate privileges. In order to stage in the complete file to the node, we leveraged the existing CMSSW callout to a curl executable. Two additional binaries were deployed to the worker node via the glideinWMS frontend—an AWS-capable curl that generates the custom AWS HTTP authentication headers, and another that wraps the SCRAM environment configuration in order to prepend an appropriate path to the environment. In addition, a bootstrap script tied into the standard CVMFS setup commands was added to the local image.[10]

8. Performance and reliability

Table 4 shows the job failure rate as seen by the WMAgent workflow management system for the various samples produced in GEN-SIM-DIGI-RECO workflows. These rates do not account for intermediate failures that are retried by the underlying HTCondor batch system (e.g. due to node preemption), or the built-in 3-try resubmission from WMAgent itself. Except for the DY_M50 outlier (where the reason is understood, and the events

[10] https://github.com/holzman/glidein-scripts

Table 4: The number of events, jobs, and failure rates of each of the 4 large scale GEN-SIM-DIGI-RECO workflows

Sample	Events	Jobs	Failure Rate
DY_M10–50	121 M	560590	0.6%
DY_M50	65 M	185971	14%*
TTJets	197 M	1089459	0.5%
WJetstoLNu	66 M	459184	0.1%

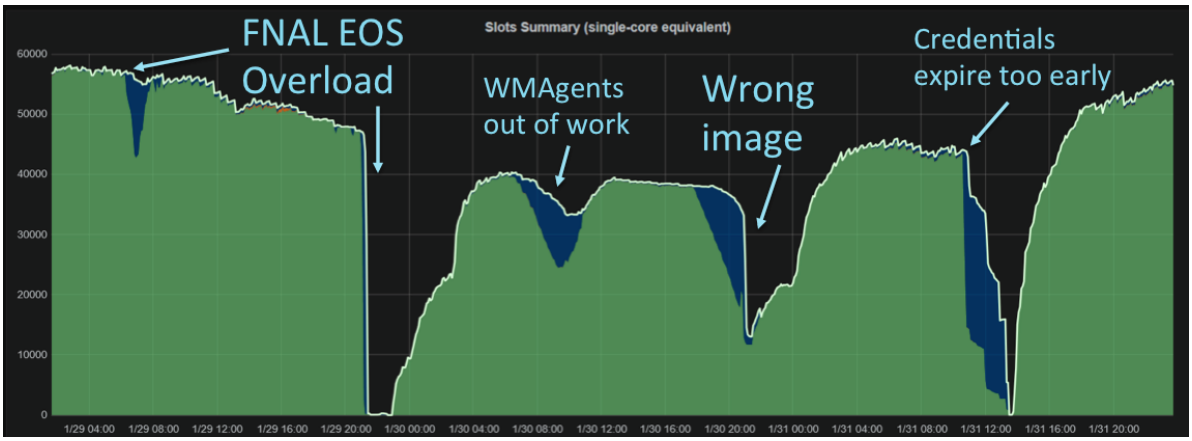

Figure 5: Count of CPU cores on AWS from January 24–31, 2016. Valleys in the distribution correspond mostly to failures on components external to AWS.

were eventually recovered), the job failure rate looks competitive or even better than what one would expect if the workflow was running at CMS-owned sites.

The reasons jobs can fail are many, but assuming the workflow itself is stable, the failures are usually related to external dependencies. Among those the most common are I/O related failures—reading input or staging output to mass storage. Figure 5 shows the effect of some of these failures on the job numbers during the time when we were trying to find a stable working point. We had problems with staging out to EOS at first, which were solved by switching from srm stageout to using xrdcp—this issue was responsible for the higher failure rate of the DY_M50 sample. There were two additional problems that affected reading pile-up from S3. The first was due to a misconfiguration in the image that caused S3 pile-up read failures; it was fixed by updating the image, shutting down the defective instances and replacing them with new ones. The second issue also caused read failures from S3—the earlier expiration of the security token used to authenticate the reads. We fixed that by only acquiring the token just before it was needed rather than when the instance was provisioned.

Apart from failure-related drops in job count, both Figure 4 and Figure 5 show that it is not trivial to keep 50,000 cores fed with jobs continuously. We had two occasions where problems with the WMAgent service prevented us from keeping the

Table 5: Wall clock and CPU time totals for CMS jobs on AWS

Sum of all jobs Wall Clock (h)	15,087,067
Sum of final jobs Wall Clock (h)	13,663,074
Sum of final jobs Cpu Time (h)	11,885,993

AWS resources busy: we couldn't keep up with the required job submission rate. These issues don't cause any failures *per se*, but have an impact on how quickly we can finish a workflow running on AWS.

Another source of inefficiencies in the system is due to the spot market. Our jobs may get pre-empted if the cost of the resource exceeds the maximum bid that CMS is willing to pay for the resource. This would not show up as a failure, since HTCondor will reschedule the job. However, it has a similar effect as failed jobs—it causes computing resources to be spent on computations for which we get no usable output. Table 5 shows the sum of wall clock and CPU time for jobs run between January 13 to February 12. The first number represents all jobs, including ones that are pre-empted and retried. The second and third are sums for only the final iteration of a job that ran to completion (including failed jobs).

From these numbers we can deduce that there was a roughly 10% inefficiency due to pre-emption losses (but the monetary losses are lower because of the way AWS bills in the case of pre-emption). The other deduction is that average CPU efficiency over all final job iterations is 87%. This is a very good number considering that our workflows run every step of GEN-SIM-DIGI-RECO sequentially on the instance and not all these steps are CPU-bound.

9. Detailed cost comparison

Cost is one of the most interesting comparisons between commercially provisioned resources and dedicated purchased computers. Historically, commercially provisioned computing has been much more expensive than regularly used purchased systems. In recent years, due to the market competition, there has been a steady decrease in the cost of commercial computing. This decrease, combined with the evolution of spot pricing as a feasible working model, has made the commercially sources of computing more cost competitive.

Fermilab attempted to estimate the cost per core hour of the CMS Tier-1 processing resources. In this calculation there are a number of very objective measurements including the cost per kilowatt of power, the initial cost of the machines, the average lifetime for computers, and the amortized cost of the computing center building. There are also several more subjective inputs, such as the actual number of people on average performing the administrative functions and the average utilization of the dedicated systems. In the cost calculations it was assumed that 3 FTE of effort were required to

Table 6: The cost per hour for one core of computing on dedicated Tier-1 resources at Fermilab and on virtualized commercial cloud resources on AWS and the $t\bar{t}$ benchmark (greater = faster). The uncertainty in the AWS cost data corresponds to one standard deviation from the daily cost per core-hour.

Site	Average cost per core-hour	$t\bar{t}$ benchmark ($t\bar{t}$ / s per core)
Fermilab CMS Tier-1	$\$0.009 \pm 25\%$ [7]	0.0163
AWS	$\$0.014 \pm 12\%$	0.0158

handle the local network and hardware administration of approximately 700 computing systems. The estimate assumes 100% utilization of Tier-1 resources; at lower utilization, the effective cost per productive CPU cycle is larger. Given the uncertainty in the subjective inputs, we estimate that the error on the per hour core cost is roughly 25%.

An important consideration when comparing the relative costs of core hours is ensuring that the work performed by each core in an hour is also comparable. CMS performed a series of benchmarks using a standard simulation workflow—the so-called $t\bar{t}$ benchmark—comparing the speed of event production on a variety of AWS instances and several generations of hardware at Fermilab. Looking at at the same number of cores used by the application, there is a spread of roughly 30%. Newer hardware generally tends to be faster for event production even if it has a lower clock speed. The spread of performance is observed at both locations and there is virtually no systematic difference between the benchmark performance on local Fermilab equipment versus AWS-hosted systems. The performance is similar, so we believe it is reasonable to directly compare the core costs.

As shown in Table 6, commercially provisioned resources are roughly 50% more expensive than dedicated well-utilized local resources, but there are some caveats. Not all workflows could be performed on AWS at this low cost rate. Workflows that have large output and incur high export charges would be more expensive, and workflows that require large random access to data not available within AWS would be less efficient and therefore more expensive. A similar test at a lower scale using the NOvA experiment measured an average cost of \$0.03 per core-hour because they needed to use larger systems and transfer data between regions. There is the potential for significant variation in cost. On the other hand, Fermilab is only this inexpensive if the resources are continuously used. One of the most attractive elements of commercially provisioned resources is that they can be dynamically provisioned. There is no difference in cost to require twice as much computing in one month and nothing in the following month. The advantages of this kind of peak scheduling will be discussed in the following section.

Given the continued evolution to lower costs in commercially provisioned computing resources and the success of this project utilizing them, it is likely that we will pursue a hybrid utilization model with contributions from both dedicated grid computing and commercially provisioned computing. There are differences in the workflows that are

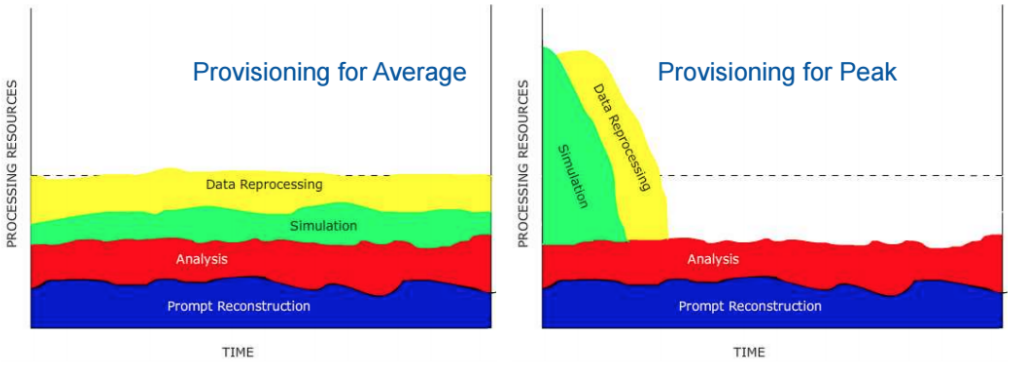

Figure 6: An illustration of provisioning for average vs. provisioning for peak.

best suited for each, so it is unlikely that one will completely replace the other, but a model with complementary resources is both desirable and cost effective.

10. Non-economic value of commercially provisioned resources

The computing resources for HEP experiments are pledged by the funding agencies of countries participating in WLCG. Because of the lead time to commission physical resources in the computing centers in the different countries, the planning process looks 18 months ahead. The HEP experiments plan and request computing resources yearly. Requests are scrutinized and eventually endorsed in a formal process by the Computing Resource Scrutiny Group (C-RSG). Resources are then made available throughout the year and it is very important that the experiments' central production teams plan for steady and continuous use during long periods of time, as shown on the left side of Figure 6. Experience from Run1 and Run2 at the LHC shows that the computing needs of experiments are not constant over time. A number of activities, such as data (re)processing, simulation data generation and reconstruction, tend to come in bursts with irregular time structure, dictated by software release, conference and data taking schedules. In the example of a conference deadline, production activities have to start well in advance to make the deadline with constant resource usage. Instead, commercial resources could enable a much more compressed processing plan starting shortly before the conference. The available elasticity of bursting resources into commercial clouds would change the way people work in large scientific collaborations and allow for shorter and more agile time schedules. The right side of Figure 6 shows this case where processing and simulation is done in burst. With resources provisioned with commercial clouds, the planning process could also be condensed. Time to provision resources is shorter because physical resources don't have to be provisioned and installed at the computing centers.

Provisioned resources like AWS may also provide a powerful source for problem recovery. In case of a problem that invalidates work already performed (a software

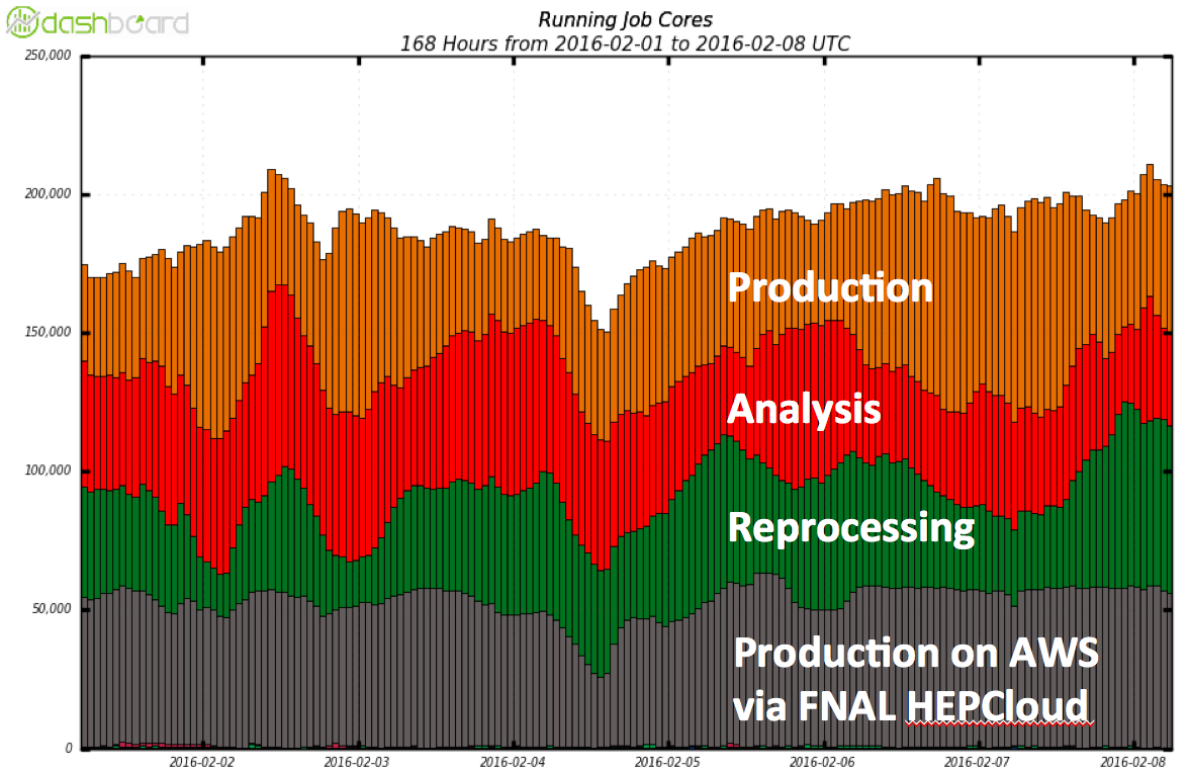

Figure 7: A comparison of the scale of processing on AWS to other global CMS activity.

problem, a systematic computing problem, or a problem of conditions) there is not sufficient excess capacity in the system to perform the work twice, without having to make difficult choices to cancel needed future work. At the same time it is not possible in the current budget environment to reserve excess capacity to recover from problems. The cloud model is interesting because it allows for the dynamic purchase of sufficient capacity to solve problems without maintaining dedicated resources in reserve. This ability to burst to a high fraction of the total CMS resources for a period of time should be seen as an useful insurance policy to recover from problems.

Specialized resources like high-memory slots and other more exotic hardware configurations might be provisioned more flexibly with commercial partners. This would allow to react flexibly and maximize physics output without long-term investment in physical hardware.

11. Looking Forward

The tests performed by Fermilab and CMS on AWS have demonstrated that it is possible to utilize dynamically provisioned cloud resources to execute many CMS workflows at large scale. As shown in Figure 7, the HEPCloud Facility was able to increase the amount of resources available to CMS by 33%. When viewed in terms of the expansion of the Tier-1 facilities, as shown in Figure 8, the effect is even larger.

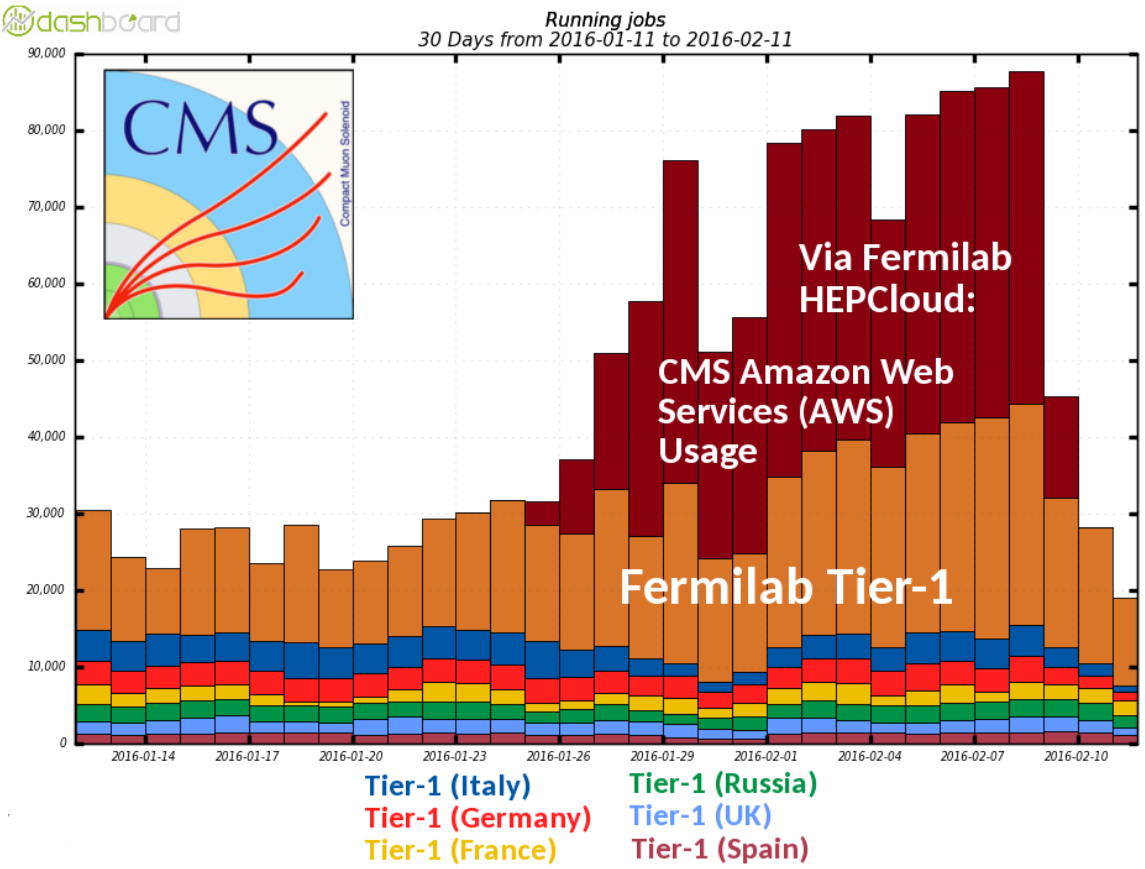

Figure 8: A comparison of the scale of processing on AWS to other CMS Tier-1 activity.

In order to operate the resources most efficiently we selected specific workflows, but we did not find any that could not be executed. If there were individual sites that wanted to satisfy their processing pledges through purchased computing services, it should be possible to maintain a similar production efficiency by enabling dedicated interface systems for grid services on top of dynamically provisioned resources.

12. Global Context of Dynamic Resources

The successful completion of real-life CMS workflows at-scale as described in this note demonstrates the potential for utilizing the HEPCloud paradigm and establishes the merits of this approach. Commercial cloud resources rented from AWS (supported through an AWS grant) were successfully integrated into the current Fermilab computing facility in a manner transparent to the experiment.

We demonstrated scalability at a level of the world-wide LHC computing scale. This data point is crucially important to extrapolate to the expected future exponential increases in computing needs, establishing that cloud provisioning is becoming a real contender for realizing these future needs.

We demonstrated that elasticity of cloud provisioning is very high, certainly

sufficient to address the bursting needs of HEP computing. We demonstrated that this approach is compatible with the operational procedures of a number of diverse experiment workflows. This approach can be used in "production mode" once the HEPCloud portal and the flexible services back-bone will be fully functional and running in operations mode. HEPCloud will then provide important new on-demand capabilities to experiment production managers; these capabilities will become a factor in the experiment resource planning.

We demonstrated that performance and efficiencies are high, comparable to and sometimes surpassing dedicated HEP resources. We measured CPU efficiency to be approaching 90 % over-all, which compares favorably with dedicated resources even given the rather complex workflows used in the demonstrator. The demonstrated reliability and high availability of the "as-a-service" approach is sufficient to serve as a back-up and insurance against eventual local outages, potentially increasing the overall robustness of the local facility.

Cost effectiveness is a complex issue, but given the increasingly competitive market of cloud providers we expect it to further increase. HEP can make use of spot market prices effectively. We saw that pre-emption caused only a 10% inefficiency overall, and the cost impact of this inefficiency was actually much lower, given the AWS spot market pricing policies. The cost comparisons assume almost 100% utilization of owned resources, a value that is rarely reached or sustained over the year, given the limited capabilities to plan ahead to the level required for full resource utilization, and the inherent "inelasticity" of experiment computing needs. As a matter of fact, the LHC utilization history of Tier-1 resources has been significantly lower, narrowing the cost gap of currently 50% to cloud-provisioned resources.

The current AWS costing model puts a premium on data transfers, making data intensive workflows a cost driver. However this may change in the future, given that the actual networking cost per unit data continues to decrease exponentially over time. HEP, through ESnet "Points of Presence" into the AWS cloud, has access to a flexible and high performance infrastructure of data access points, which should bring down the actual data transport costs to the provider. Using data transfer volume as a cost driver is part of AWS' busyness model today, but it is at least conceivable that future cost models for large-scale clients like HEP could de-emphasize data fees while still providing enough margin for cloud providers to be profitable.

The demonstrator described in this note is just the beginning of making HEPCloud a dependable part of the Fermilab and HEP computing infrastructure for the LHC, the intensity frontier and the neutrino program. For the LHC in particular, the new capabilities of on-demand resources and resource elasticity provided by HEPCloud are significant enough to partially outweigh the larger cost per CPU hour, making HEPCloud provisioning of LHC resources a new and important ingredient to the ecosystem of computing resources. HEPCloud adds commercial or community cloud resources to the predominance of owned resources, augmenting opportunistic resources across OSG, and future HPC and supercomputing center resources to the mix. This

approach will help HEP facilities move away from standalone, silo-ed solutions.

We expect the new capability of reliable and robust on-demand provisioning of HEPCloud to significantly impact future resource planning for the LHC and the rest of the HEP computing program. For the next year of computing resource planning, on-demand capabilities will lower the need for owned resource on the floor, lowering the need for over-provisioning as a strategy to deal with peak demands. The corresponding decrease in investment cost for owned resources will provide some flexibility for on-demand capacity. The exact balance for the coming resource years will need to be carefully determined. In the meantime, making the HEPCloud portal a robust and fully supported piece in the US HEP computing landscape will be of high priority and importance.

We anticipate that the HEPCloud portal concept will provide a means for all laboratories to provide shared resources in the ecosystem, resulting in a large pool of offerings for compute, archival capabilities, database services, data management, etc., potentially linking all US HEP computing.

13. Conclusions

The HEP experimental program continues to evolve, and will require computing capacity in excess of current and future on-premises resources. It is key that we acquire the capability to expand HEP facilities beyond what is on the local data center floor. A sensible target which will yield the most benefits is to leverage the industry trends in cloud computing. The HEPCloud Facility concept provides a portal to these computing resources as a transparent layer for the users, offloading the decisions of when and how to acquire off-premises resources to the Facility.

We deployed an implementation of the HEPCloud Facility using glideinWMS technology, with the goal of serving a number of different use cases and experiments. The goal of the CMS use case was to enable the execution of a physics workflow that would add significantly to not only their overall global resource consumption but also generate useful analysis results for the experiment. For a full simulation workflow (from event generation to physics reconstruction), over 15 million hours of computing was consumed, simulating more than 500 million events. The steady-state cost came to $1.4 \pm 12\%$ cents per core-hour, which is not much larger than the estimated $0.9 \pm 25\%$ cents per core-hour for the Fermilab data center. The NOvA experiment also executed a smaller-scale workflow on AWS (at a cost of 3 cents per core-hour), demonstrating the ability of the HEPCloud Facility to serve different user communities and exercising a different processing-to-data ratio.

From this work, we have shown that commercial cloud resources can be acquired at large scales for costs that are larger than, but comparable to, the cost of procuring and deploying similar resources on-site. Given the large year-over-year increases in the size of cloud computing industry-wide and the potential economies of scale, it is conceivable that the steady-state computing costs could approach or even undercut the price of

procuring physical equipment. Beyond the comparison of steady-state costs, the needs and demands of the scientific community are not flat with respect to time, but have a structure and time-dependence. Having the ability to pay for only the resources that are used gives a large amount of flexibility and can increase the efficiency and cost effectiveness overall. On the other hand, not all workflows are well-suited to running off-site. Despite the international proliferation of high-bandwidth networks, scientific workflows that are very data-intensive may be better matched to executing on local resources near storage. It is clear that a hybrid approach, the HEPCloud Facility—capable of provisioning both on-premises and off-premises resources and aggregating them into a single virtual facility—will give the most flexibility and gives the scientific community the best chance to meet the ever-growing needs of its users.

References

[1] Gartner Group 2015 Magic Quadrant for Cloud Infrastructure as a Service, Worldwide *https://www.gartner.com/doc/reprints?id=1-2G45TQU&ct=150519&st=sb*

[2] Amazon Web Services 2016 AWS Offers Data Egress Discount to Researchers *https://aws.amazon.com/blogs/publicsector/aws-offers-data-egress-discount-to-researchers/*

[3] Fermilab 2015 Fermilab Request For Proposals for Cloud Resources and Services *http://cd-docdb.fnal.gov/cgi-bin/ShowDocument?docid=5735*

[4] Mhashilkar P, Tiradani A, Holzman B, Larson K, Sfiligoi I and Rynge M 2014 Cloud Bursting with GlideinWMS: Means to satisfy ever increasing computing needs for Scientific Workflows *J. Phys.: Conf. Ser.* **513** 032069

[5] Timm S, Garzoglio G, Mhashilkar P, Boyd J, Bernabeu G, Sharma N, Peregonow N, Kim H, Noh S, Palur S and Raicu I 2015 Cloud Services for Fermilab Stakeholders *J. Phys.: Conf. Ser.* **664** 022039

[6] Wu H, Ren S, Timm S, Garzoglio G and Noh S-Y 2015 Experimental Study of Bidding Strategies for Scientific Workflows using AWS Spot Instances *8th IEEE Workshop on Many-Task Computing on Grids and Supercomputers (MTAGS)*

[7] Fuess S, unpublished communication